Project 0: Word ladders

Due: 1 February 2022

A word ladder is a sort of puzzle wherein two words (of equal length) are given, and the solver has to find a chain of words to "connect" them by only changing one letter at a time. For instance, given the words CAT and DOG, one valid ladder would be

CAT
RAT
RAG
BAG
BOG
DOG

though that's not the shortest possible one.

This project is sort of a warm-up, to give you extra programming practice and confirm that you're up to speed on your general programming skills. You'll implement a program that solves word ladders.

Interface

The program should get three pieces of information as command line arguments: the start word, the end word, and the name of a dictionary file. It should then print (to standard output) either a valid word ladder, or a message indicating that none exists. It may print additional debugging information to standard error.

Dictionary files

A dictionary file is a text file containing one word per line. Some dictionary files contain words with apostrophes or spaces, or words that begin with a capital letter, but you should ignore these.

Most modern unix-type systems include a suitable dictionary file in `/usr/share/dict/` (on our lab systems it is in `/usr/share/dict/american`).

Checkpoint

For the checkpoint (next Thursday), you should have a program that:

- gets the three command line arguments,
- verifies that the start and finish word are of the same length (and exits with a clear message if not),
- reads the dictionary file, and
- prints out all the valid words that differ from the start word by exactly one letter.

For instance, if the start word was "slump" the program would find "clump", "flump", "plump", "stump", and "slurp", at least using the dictionary on the lab machines. Note that you can (should!) write a very short dictionary of your own to test with, in addition to making sure your code works with the system's dictionary.

Final version

A full-credit final version will be a complete, non-buggy, working implementation of the word ladder problem specified above, TOGETHER WITH convincing proof that it is correct. The "proof" should consist of test cases (in whatever format is convenient to you) to illustrate various situations, including both input and expected results.

Remember that there need to be clear instructions on how to run it in general as well as how to run each/all of your tests and quickly verify that they ran correctly (and which rubric items each one corresponds to). Having complete and correct documentation is an easy 15 points, but if your documentation omits important info or tells me the wrong thing, you'll get less than full credit there.

After checkpoint work (15 points) and documentation (15 points), there remain 70 points in the rubric, which will be awarded according to the table below. Under each score, I show (for your convenience) the total cumulative points if you get that item plus *all* the previous points, and the letter grade this corresponds to. It is arranged roughly in the order I suggest you attempt

them, with the earlier ones being easier or enlightening with respect to the later ones, but you can in general get points for the later ones (if they work) without getting the earlier ones.

NOTE: if your code doesn't compile, or immediately crashes when it's run, you will get zero of these points. Don't let this happen to you!

ALSO NOTE: if it runs but it takes "forever" (more than, say, a couple minutes), I'll terminate the process; if you think your program is prone to that and you worry about losing points, make sure to document things like how long it usually takes and why your program should get various points.

Score	Description
10 (40/D−)	Runs on all valid inputs without crashing (and rejects invalid inputs), and either prints all valid words adjacent to start (per the checkpoint) or, for *at least some* word pairs that differ by more than one character, prints a valid ladder (per the final spec).
5 (45/D)	Works with correct output for all valid length-1 ladders (i.e. the start and finish are both valid and differ by one character).
5 (50/D+)	Identifies at least some words at distance ≥ 2 from the start or finish via some valid intermediate word(s). *
5 (55/C−)	Finds and prints (per the output spec) valid ladders for *at least some* word pairs that differ by more than one character (even if it only works for length-2 ladders).
10 (65/C+)	Finds and prints a valid ladder, not necessarily the shortest, in *all* cases where the word pair differs by only two characters and some path exists.
10 (75/B)	Finds and prints a valid ladder, not necessarily the shortest, in *all* cases where a valid ladder exists.
5 (80/B+)	Tracks, at least sometimes, words that have already been explored, and avoids re-exploring them. *
10 (90/A)	Correctly identifies all cases where no valid ladder exists and prints a message to that effect, and never gets stuck in an infinite loop.
10 (100/A+)	Finds *the shortest* valid ladder, in all cases where a valid ladder exists. **

* Note that these rubric points are not particularly "visible" from the expected final output, but are implied by various later points. If you don't have the later points and want to claim these, make sure you explain how

I'd know you're doing it (and what to look for in your debugging output)!

** How do you *know* it's the shortest? How should *I* know? Consider what might constitute "convincing proof" on this one, and make sure you have it in your documentation. (If there's reference to your code, make sure to include specifics and line numbers....)

Handing in

Hand it in as `proj0` using the handin program (see general project notes for instructions).